UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DISEÑO DE REDES DE CÁMARAS INTELIGENTES UTILIZANDO SMARTPHONES

Fernando Lahoz Seguido
Tutor: Juan Carlos San Miguel Avedillo
Ponente: José María Martínez Sánchez

Julio 2017

DISEÑO DE REDES DE CÁMARAS INTELIGENTES UTILIZANDO SMARTPHONES

Fernando Lahoz Seguido
Tutor: Juan Carlos San Miguel Avedillo
Ponente: José María Martínez Sánchez

Video Processing and Understanding Lab
Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2017

Trabajo parcialmente financiado por el Ministerio de Economía y Competitividad del Gobierno de España bajo el proyecto TEC2014-53176-R (HAVideo) (2015-2017)

Resumen

Este Trabajo Fin de Grado (TFG) presenta una herramienta software para controlar de forma remota una red de cámaras inteligentes utilizando smartphones Android. Dicha red compone un sistema de videovigilancia casero que consta de tres partes diferenciadas: teléfonos móviles (cámaras) y aplicaciones cliente y servidor.

Este sistema será implementado aprovechando las posibilidades que nos ofrece Android a la hora de controlar las diferentes configuraciones de captura y envío de imágenes. También será utilizada la herramienta de procesamiento de imágenes OpenCV a la hora de visualizar dichas imágenes en el cliente y aplicar un algoritmo de detección de personas sobre las mismas cuando sea necesario.

La compleja gestión de la red de cámaras y usuarios, que deben ser tratados de forma diferente, será llevada a cabo por un servidor, que será el centro del sistema, encargado del protocolo de comunicaciones entre usuarios y cámaras.

Una vez terminado el desarrollo, se realizaron pruebas experimentales de rendimiento con diferentes configuraciones de captura y número de cámaras simultaneas para sacar valiosas conclusiones acerca de las limitaciones de la arquitectura propuesta para este sistema de videovigilancia.

Palabras clave

Videovigilancia, Android, sistema, control remoto, sistema distribuido de múltiples cámaras, OpenCV.

Abstract

This final degree project presents a software tool to remotelly control a smart multicamera network using Android smartphones. This network is composed by a homemade videosurveillance system which consists of three different modules: smartphones (cameras), server and client applications.

This system will be implemented by taking advantage of Android possibilities when controlling different camera capturing configuration and sending images. The processing images tool OpenCV will also be used to visualize these pictures in the client aplication and apply a people detection algorithm when it's requested.

The complex managing of the camera and user network must be treated differently, it will be done by a server, which will be the system center in charge of the comunication protocol between users and cameras.

Once the development is finished, experimental performance tests will be done with different camera capture configurations and variating the number of simultaneous cameras to come to the conclusion of the proposed architecture's limitations for this videosurveillance system.

Keywords

Videosurveillance, Android, system, remote control, distributed multicamera system, OpenCV.

Agradecimientos

Me gustaría agradecer a mi tutor del TFG, Juan Carlos San Miguel Avedillo, su ayuda, tiempo y disponibilidad durante el año de trabajo que me ha llevado a finalizar este proyecto de fin de grado. En especial por su esfuerzo en las últimas semanas de acabar el proyecto para ayudarme a dar ese empuje final, cuando pueden fallar las fuerzas, poniendo a mi disposición una sala entera en el laboratorio, conocido como VPULab, con la tecnología necesaria para llevar a cabo todos los experimentos necesarios.

Quisiera agradecer también a mi familia el apoyo recibido y en especial a mi madre por estar ahí, siempre que he necesitado su ayuda, todos estos años de estudios hasta haber logrado mis objetivos. Me acuerdo con nostalgia de esos momentos con mi hermana Cristina, que me dejaba el teléfono a regañadientes para poder hacer pruebas con múltiples cámaras cuando aún no tenía varios móviles para realizar los experimentos.

Agradecer también a mis compañeros de la EPS, que siempre que les he pedido ayuda me la ofrecieron sin rechistar. Y en especial a mi compañero y amigo Pablo Sala, que al haber llevado a cabo trabajos de fin de grado similares, pudimos ayudarnos mutuamente.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

La videovigilancia (Figura 1.1) es un área en expansión, que existe para satisfacer una necesidad básica como es la seguridad, ya sea personal o material. Desde tiempos inmemoriales, el ser humano se ha preocupado por la seguridad, la vigilancia y la protección de sus propiedades. Hoy en día se hace mucho más sencillo gracias a complejos sistemas hardware de videovigilancia y algoritmos de detección de objetos, movimientos y personas.

Una de las motivaciones de este proyecto es hacer más accesible un sistema de este tipo, utilizando las ventajas que ofrece un teléfono móvil moderno (smartphone Android), al alcance de casi cualquier persona hoy en día, un ordenador que sirva como servidor (sin unos altos requerimientos), y un dispositivo con la aplicación cliente para poder acceder al contenido visual o los datos obtenidos por los móviles (de ahora en adelante cámaras) así como controlar multitud de preferencias por control remoto, como por ejemplo la resolución, los FPS (imágenes por segundo), etc.

Los motivos más significativos para llevar a cabo este proyecto son los siguientes:

- Las cámaras y el servidor pueden ser transportados de un lugar a otro fácilmente, sin necesidad de alimentación eléctrica externa, ya que se utilizan las baterías de los propios teléfonos.

- Es mucho más barato que cualquiera de los actuales sistemas de videovigilancia profesionales.

- Portabilidad muy alta, ya que el código es fácilmente integrable en otros sistemas operativos.

Figura 1.1: Sistema de videovigilancia

- Las posibilidades de control remoto, versatilidad y autonomía que ofrece un smartphone a día de hoy.

1.2. Objetivos

El objetivo final de este proyecto es desarrollar un sistema de videovigilancia compuesto por una red de cámaras inteligentes utilizando dispositivos móviles Android, que permitan su control e interacción de forma remota.

El proyecto va a estar dividido en tres módulos claramente diferenciados. Por un lado estará la aplicación móvil, que será la encargada de realizar la captura de imágenes (previamente configurada). Por otro lado se encontrará el servidor, con un protocolo de comunicaciones robusto, capaz de controlar las cámaras y de enviar comandos a dichas cámaras o recibirlos por parte del tercer módulo, la aplicación cliente, encargada de realizar peticiones a las cámaras a través del servidor.

Tomando como punto de partida estas premisas, para llevar a cabo este proyecto, será necesario que se cumplan los siguientes objetivos:

- Estudiar el arte actual para comparar propuestas actuales en sistemas de videovigilancia caseros y profesionales.
- Estudiar las posibilidades que ofrece Android para el control de la cámara del dispositivo.
- Diseñar un sistema multicámara de videovigilancia, portable, con un servidor central y capaz de soportar múltiples clientes simultáneos.

- Planear y definir un protocolo de comunicaciones completo entre las cámaras, el servidor y la aplicación cliente.
- Implementar el software correspondiente al teléfono móvil Android que sea capaz de acceder a la cámara, configurarla, capturar imágenes y enviarlas.
- Desarrollar el software para el servidor de forma robusta y prestando especial atención a la eficiencia.
- Implementar una aplicación cliente que sirva para poder observar las imágenes obtenidas por las cámaras en tiempo real y controlar las diferentes configuraciones de las cámaras Android mediante el uso de los comandos ya definidos en el protocolo de comunicaciones.
- Integrar OpenCV (ver anexo A) en el algoritmo de detección de personas[1] en el sistema para darle una utilidad práctica al proyecto.
- Realizar experimentos de rendimiento a diferentes resoluciones, FPS y cámaras.

1.3. Estructura del documento

La memoria consta de los siguientes capítulos:

- Capítulo 1. Introducción.

  Capítulo de introducción sobre la motivación y objetivos del TFG.

- Capítulo 2. Estado del arte.

  Capítulo que aporta una visión general a los sistemas actuales en el campo de la videovigilancia y control remoto de cámaras y *smartphones*.

- Capítulo 3. Diseño del sistema.

  En este capítulo se aborda el sistema desde fuera, prestando atención al diseño y el funcionamiento global.

- Capítulo 4. Implementación.

  En este capítulo se describe el sistema de forma detallada, prestando atención a los detalles de comunicaciones a bajo nivel, la arquitectura hardware y software, así como otros detalles de la implementación realizada.

- Capítulo 5. Trabajo experimental.

  Este capítulo lo compone una serie de pruebas, experimentos, comparativas y gráficas, en base a lo implementado.

- Capítulo 6. Conclusiones y trabajo futuro.

  Capítulo de cierre que trata de recopilar ideas para una posible ampliación futura, así como conclusiones obtenidas de los experimentos realizados.

Al final del documento, después de la bibliografía, se encuentran los siguientes anexos:

- Anexo A. Desarrollando en OpenCV.

  Explicación general de la herramienta para el procesamiento de imágenes *OpenCV* y su integración en Android.

Capítulo 2

Estado del arte

Este capítulo hace un análisis de los diferentes sistemas de videovigilancia y control remoto. A continuación, diferentes comparativas entre los sistemas actuales de videovigilancia y control remoto con este proyecto nos darán una visión clara de los puntos clave a tratar en cualquier proyecto de este tipo.

Los sistemas más extendidos para realizar la integración de un sistema de videovigilancia profesional son los compuestos por una cámara IP y un software que saque provecho de las mismas, normalmente solo compatible con las cámaras IP de su marca.

En este capítulo nos centramos en analizar los puntos clave de los sistemas de videovigilancia que usen un teléfono inteligente Android como cámara IP.

2.1. Sistemas de videovigilancia en Android

2.1.1. AtHome Camera - Home Security

Es uno de los sistemas de videovigilancia remotos más descargados de la lista de Google Play (la tienda oficial de Android). Permite usar un teléfono Android con la app instalada como una cámara accesible de forma remota. Posee características muy interesantes como notificaciones por email ante detección de movimiento y dispone de grabación programada de vídeo (Figura 2.1).

2.1.2. Video Monitor - Surveillance

Otro sistema de videovigilancia usando móviles Android muy popular. Permite usar un teléfono Android como una cámara accesible remotamente. Como características destacadas tiene: sensor de movimientos con notificación por email, seguimiento de la localización por GPS y grabación de vídeo para verlo en diferido (Figura 2.2).

Figura 2.1: AtHome Camera Home Security

2.1.3. i-Security

Otro popular sistema de videovigilancia remoto en Android. Funciona de forma similar a sus competidores. Dispone de sensor de movimiento y puede monitorizar hasta 64 cámaras simultáneamente (Figura 2.3).

2.1.4. Ivideon

Uno de los más descargados sistemas de videovigilancia en móviles Android. Dispone de notificaciones por email ante movimientos y cambios de sonido, requiere muy poco ancho de banda (funciona con una conexión 2G), permite transmitir el vídeo para incrustarlo en portales web (Figura 2.4).

2.1.5. Alarm.com

Otro sistema de videovigilancia para Android. Este destaca por tener un mayor control sobre el propio sistema de forma remota, permitiendo armar o desarmar el sistema, controlar el termostato, bloquear y desbloquear puertas, controlar la iluminación, etc (Figura 2.5).

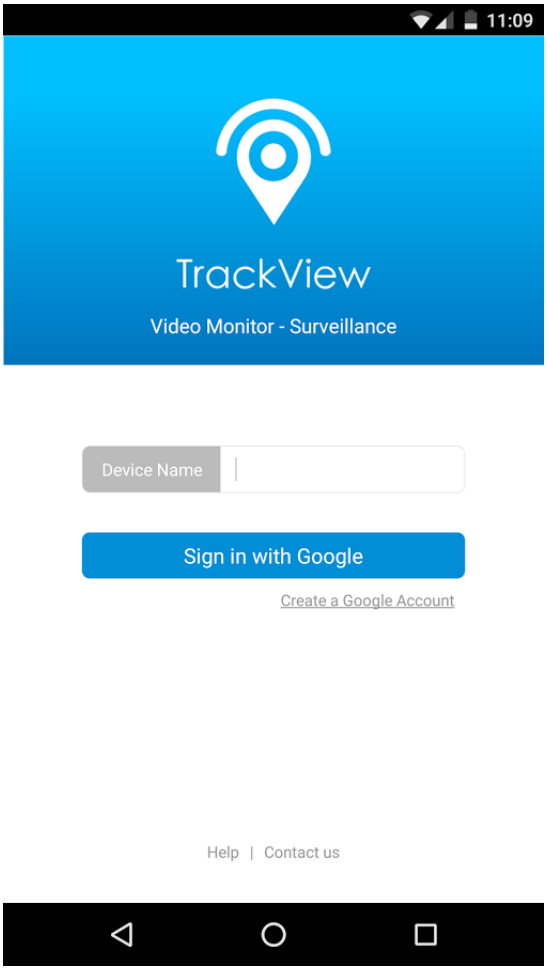

Figura 2.2: VideoMonitor - Surveillance

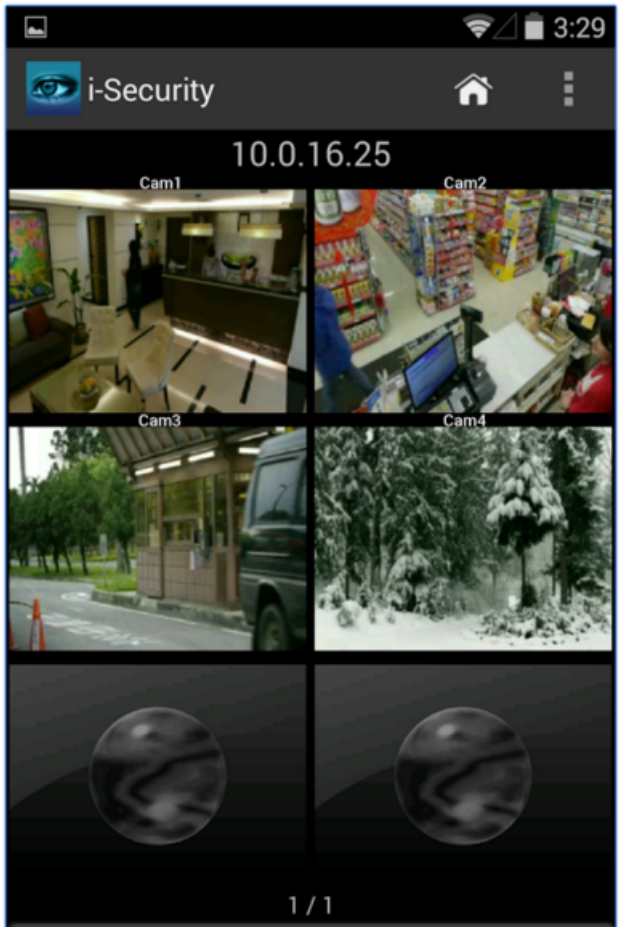

Figura 2.3: i-Security

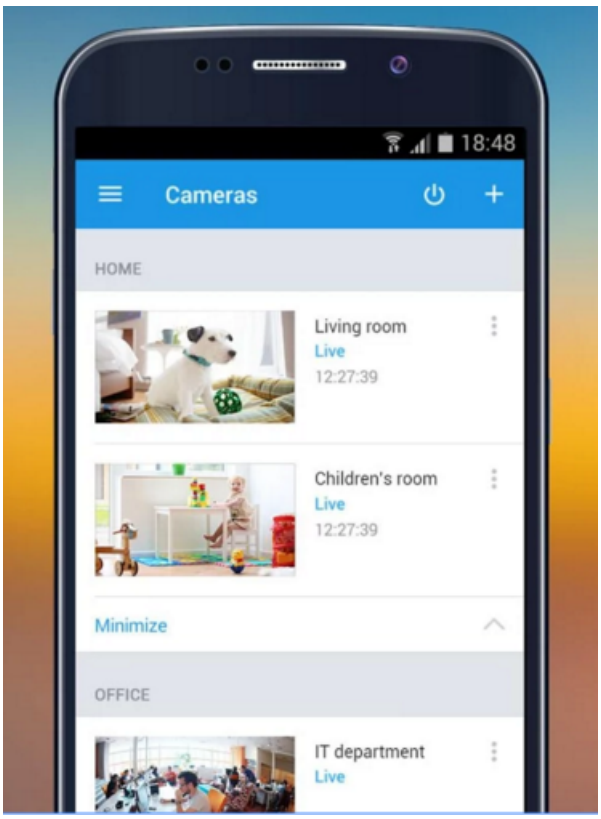

Figura 2.4: Ivideon

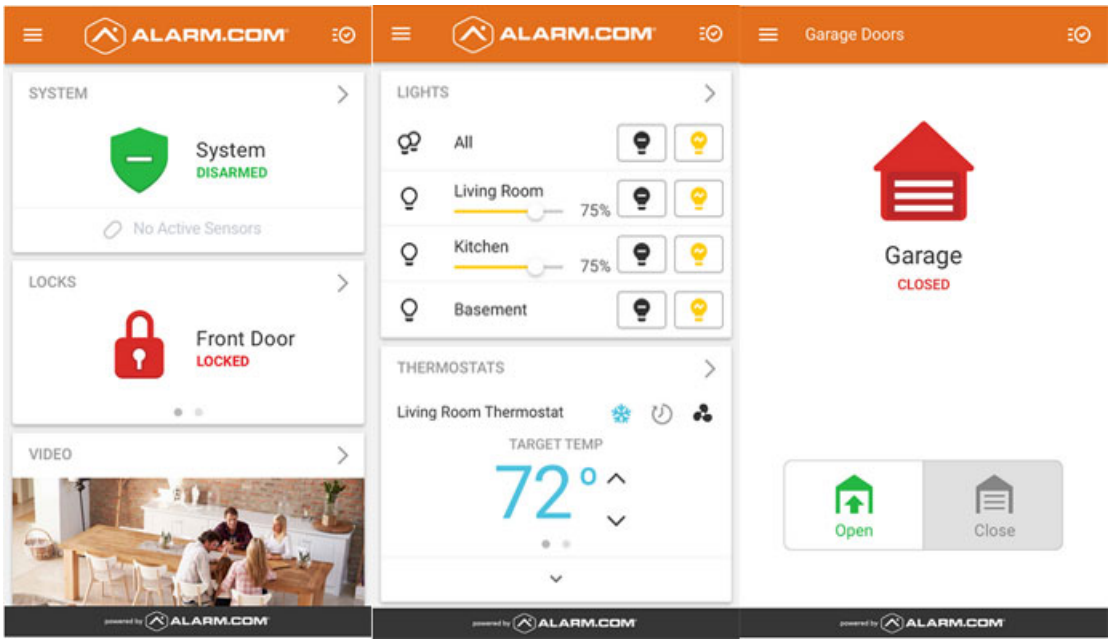

Figura 2.5: Alarm.com

	Notificación email	**GPS Tracking**	**Sonido**	**Control de dispositivos**
AtHome	Sí	No	No	No
Video Monitor	Sí	Sí	No	No
iSecurity	No	No	No	No
Ivideon	Sí	No	Sí	No
Alarm.com	No	No	No	Sí

Tabla 2.1: Comparativa entre sistemas de videovigilancia en Android

2.2. Comparativa entre sistemas actuales

Al momento de comparar los sistemas actuales de videovigilancia debemos elegir las características más importantes a tener en cuenta. Para ello, se ha elaborado una tabla (ver Tabla 2.1) que podremos usar para sacar conclusiones acerca de los sistemas de videovigilancia más populares del mercado, reflejando los puntos claves comunes a todos ellos y algunas características únicas de cada uno de ellos.

2.3. Conclusiones

La competencia entre los sistemas de videovigilancia que utilizan teléfonos inteligentes Android como cámaras es muy fuerte. Parece que los sistemas tradicionales de transmisión de vídeo se quedan cortos ante la cantidad de posibilidades que ofrecen los sensores (luz, proximidad, micrófono, cámaras, temperatura...) y la conectividad (wifi, bluetooth, infrarrojos, NFC...) de un dispositivo Android. Es por eso que empiezan a aflorar los sistemas de control remoto del hogar, aplicaciones que permiten controlar toda clase de dispositivos en el hogar, como la iluminación, bloqueo de puertas, monitorización mediante cámaras, etc.

La conclusión que se saca de este estudio de los sistemas actuales es que hay que enfocarse en ofrecer el máximo de funcionalidades posibles gracias a lo bien dotados que están los teléfonos inteligentes. Por ello, centramos el proyecto en, además de ofrecer vídeo en tiempo real con múltiples cámaras, ser capaces de controlar todo tipo de parámetros de la captura, para adecuarla a nuestras necesidades.

Capítulo 3

Diseño del sistema

Este capítulo contiene los requisitos del sistema, su diseño y algunos conceptos sobre el sistema desarrollado, sobretodo en lo referente a la comunicación entre los diferentes módulos de la aplicación y el flujo de ejecución.

3.1. Análisis de requisitos

A lo largo de esta sección, serán analizados los requisitos que se deberán cumplir en la etapa final del proyecto. Para clasificar dichos requisitos se dividirán en dos tipos: requisitos funcionales y requisitos no funcionales.

3.1.1. Requisitos funcionales

- La red de cámaras podrá estar compuesta por múltiples cámaras coordinadas por un mismo servidor local o remoto.
- Las cámaras del sistema serán configurables de forma remota a través de una aplicación cliente que enviará las ordenes a ejecutar al servidor al que están las cámaras conectadas.
- El único dato que será necesario conocer tanto en el lado de las cámaras como en el de los clientes es la dirección IP y puerto del servidor.
- El reconocimiento de personas podrá ser activado en cualquier momento desde la aplicación cliente.
- Será ajustable la resolución y los FPS en todo momento desde la aplicación cliente.

- En cualquier momento se podrán añadir o quitar cámaras del sistema sin necesidad de reiniciar o configurar nada en el servidor, que deberá adaptarse automáticamente a los cambios.
- Protocolo de comunicaciones robusto que permita la comunicación de instrucciones del cliente al servidor, del servidor a las cámaras, y de vuelta desde las cámaras al cliente pasando por el servidor intermedio.

3.1.2. Requisitos no funcionales

- Rendimiento: el rendimiento en un sistema de transmisión de vídeo de alta calidad en tiempo real es un factor crítico, y deberá ser el suficiente para permitirnos realizar dicha tarea de forma eficiente (ver sección 5.4).
- Disponibilidad: se requiere que el sistema sea robusto y esté funcionando de forma continua y sin caídas (ver sección 5.3).
- Mantenibilidad: se desarrollará el proyecto de una forma organizada, modular y realizando una API de programación para tener acceso a las funciones Android de forma simple (ver sección 4.4).
- Documentación: Con el presente documento se ofrece una extensa documentación para el que no esté familiarizado con sistemas de videovigilancia, ni la instalación de los mismos y pueda instalar, configurar y usar este sistema de videovigilancia casero sin demasiada dificultad (ver sección 5.1).
- Usabilidad: Se tratará que la aplicación cliente sea lo más práctica e intuitiva posible (ver subsección 4.3.3).

3.2. Resumen del sistema

El sistema de videovigilancia está compuesto de tres módulos (Figura 3.1): cámaras, servidor y aplicación cliente.

Las cámaras serán los móviles Android encargados de conectarse al servidor y permanecer a la escucha de peticiones que vengan del mismo para realizar acciones como configurar parámetros de la grabación, controlar las conexiones wifi o bluetooth, enviar información sobre el estado actual, comenzar la captura de imágenes y enviar dichas imágenes al servidor.

El servidor es el módulo intermedio entre la aplicación cliente y las cámaras. La aplicación cliente es la encargada de enviar comandos al servidor, que será el

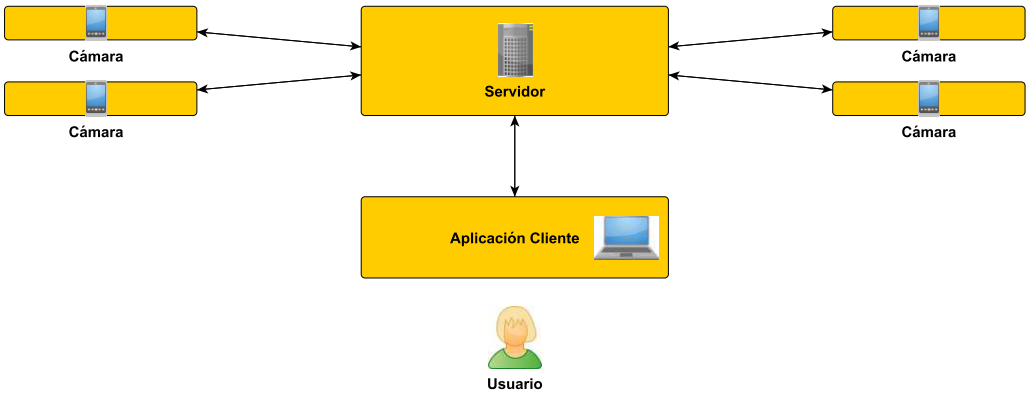

Figura 3.1: Diseño global del sistema

encargado de adaptar dichos comandos a peticiones a las diferentes cámaras, obtener sus respuestas y mandar dicha información a la aplicación cliente.

La aplicación cliente es la encargada de enviar los comandos a las cámaras a través del servidor. El hecho de tener un servidor intermedio hace que sea mucho más sencilla de implementar la comunicación con múltiples cámaras de manera simultanea.

3.3. Comunicación entre módulos del sistema

En esta sección se va a explicar el diseño de las comunicaciones entre los módulos ya descritos del sistema. Se puede visualizar dichas comunicaciones en la figura 3.2.

3.3.1. Comunicación entre cámaras Android y servidor

Los teléfonos móviles Android, es decir, las cámaras, realizan una conexión inicial al servidor en la que se las asigna un identificador único. Después permanecerán inactivas a la espera de recibir comandos. Los comandos que se intercambian entre las cámaras y el servidor son los siguientes:

- *Connect*: Conecta la cámara al servidor, quien le asigna un identificador único.
- *Get current*: Obtiene información de los parámetros de captura, procesador y estado de las conexiones inalámbricas.
- *Set current*: Define un nuevo valor para alguno de los parámetros configurables ya mencionados.
- *Start*: Inicia la captura de imágenes usando los valores configurados previamente.

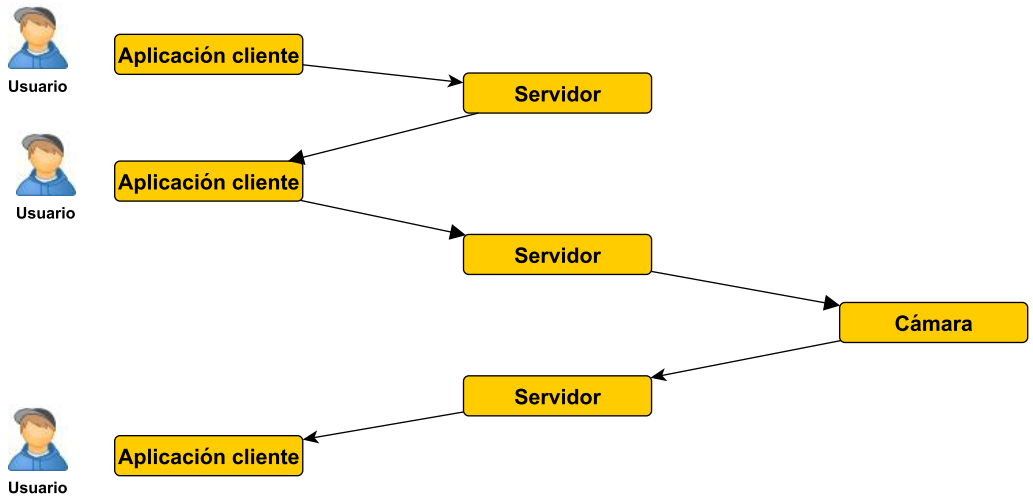

Figura 3.2: Flujo de comunicaciones habitual

- *Stop*: Detiene la captura de imágenes iniciada por el comando anterior.
- *Get Frame*: Una vez iniciada la captura de imágenes con el comando start, se puede realizar esta acción, que sirve para obtener la última imagen disponible.

3.3.2. Comunicación entre servidor y clientes

La aplicación cliente es la encargada de enviar las acciones solicitadas por el usuario a las cámaras a través del servidor. De este modo, los comandos de la sección anterior podrán ser enviados al servidor, especificando el identificador de la cámara concreta a la que se quiere enviar.

Los detalles sobre la implementación de dichos comandos y el protocolo de comunicaciones en detalle será explicada en el siguiente capítulo.

Capítulo 4

Implementación

Basado en los requerimientos descritos en el capítulo anterior, este capítulo introduce la arquitectura hardware y software del sistema desarrollado. Después, pasaremos a explicar la implementación y la interconexión de los tres módulos que componen el sistema de videovigilancia. Para finalizar, detallamos la funcionalidad de la API Android, el protocolo de comunicaciones y ejemplos de uso prácticos.

4.1. Arquitectura hardware

En esta sección detallamos la arquitectura hardware del sistema. Esto involucra a las diversas tecnologías que intervienen en el proceso de captura de imágenes, procesamiento de las mismas y su envío.

- **RAM.** Es fundamental en el sistema ya que se utiliza para almacenar las imágenes de forma temporal hasta ser mostradas y liberadas posteriormente. La razón de almacenar imágenes en memoria en vez de en disco es la necesidad de emitir dichas imágenes en tiempo real, lo cual sería muy difícil usando la velocidad de un disco dura estándar ya que el costoso tiempo de escribir y leer de disco haría imposible mantener unos FPS suficientemente altos.

- **CPU.** Las imágenes capturadas por la cámara tienen un tipo concreto que no se adapta al sistema. Para que estas imágenes sean compatibles con el sistema se deben convertir del formato de captura YUV420 por defecto en Android a RGB y posteriormente comprimirlo a JPEG. Esta tarea debe ser realizada por la CPU del móvil antes de su envío por la red. Por lo tanto, es vital que las cámaras cuenten con un procesador adecuado que permita realizar estas tareas tardando el menor tiempo posible para mantener un ritmo de imágenes por segundo que de sensación de movimiento.

- **Tarjeta de red.** La tarjeta de red permite la comunicación vía Internet del sistema. Es necesario un ancho de banda suficiente dependiendo del número de cámaras conectadas simultáneamente, la resolución de las mismas y los FPS que se quieran obtener. En el capítulo 5 analizamos cuanto sería necesario para trabajar en diferentes condiciones.

4.2. Arquitectura software

Como vimos en el capítulo anterior, el sistema de videovigilancia está compuesto por tres módulos: cámaras, servidor y aplicación cliente. Las cámaras capturan las imágenes, realizan su conversión al espacio de color RGB, después comprimen los datos al formato JPEG y realiza el envío de las mismas. El servidor se encarga de hacer de intermediario entre las cámaras y la aplicación cliente, de modo que cuando un usuario solicita una acción es este módulo el encargado de analizar dicho comando, enviarlo a las cámaras que hagan falta, esperar por la respuesta y contestar al usuario de nuevo. La aplicación cliente es un módulo que nos otorga funcionalidad como usuarios. Desde él podemos enviar acciones de consulta o ejecución a las cámaras a través del servidor especificando el identificador de las cámaras que queramos.

4.3. Módulos

En esta sección se habla más detalladamente acerca de la implementación de cada uno de los módulos que componen este sistema de videovigilancia.

4.3.1. Cámara Android

En el desarrollo de la aplicación móvil que permita la captura de imágenes, su procesamiento y envío se ha utilizado el entorno de desarrollo Android Studio para realizar la implementación Java de la app con las librerías de procesamiento de imágenes de OpenCV incluidas en el ejecutable generado como producto final.

Se ha desarrollado una aplicación intuitiva, fácil de usar, pero personalizable en las opciones que ofrece. Dispone de una interfaz gráfica de usuario que informa en todo momento del estado del dispositivo móvil, en concreto del estado de la cámara y de la conexión con el servidor remoto. Además de funcionalidad visual también dispone de una compleja organización que ofrece desde una API con muchas funcionalidades relativas a la cámara, la conexión y el tratamiento de imágenes hasta una gestión inteligente de los recursos del teléfono para responder rápidamente a los comandos que se reciban de un usuario, a través del servidor, creando dos hilos para su ejecución

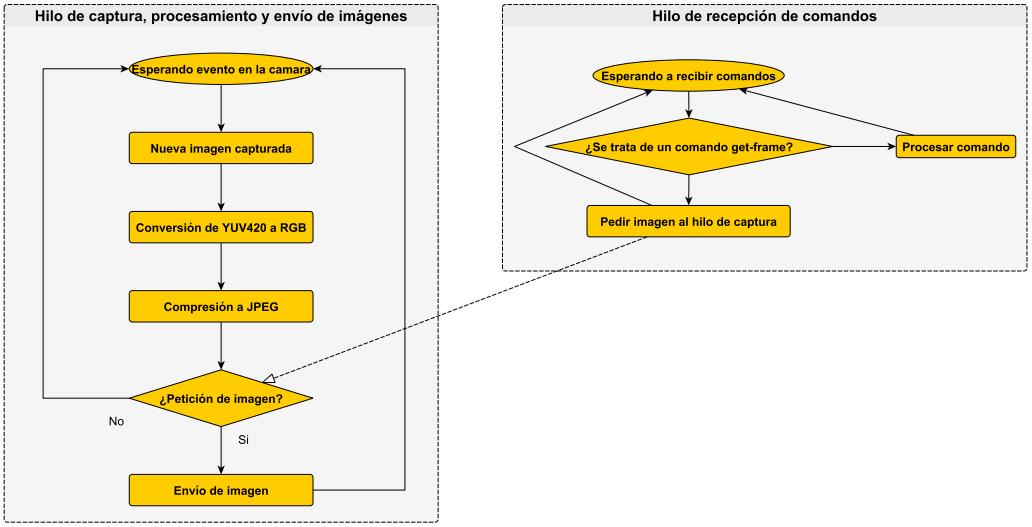

Figura 4.1: Procesamiento y envío de imágenes en la cámara

en paralelo: uno que gestiona todo lo relativo a la conexión con el servidor y otro que gestiona lo relativo a la captura, conversión y envío de imágenes. Esto puede observarse claramente en la imagen de la Figura 4.1.

A continuación se detalla cada uno de los componentes de la aplicación móvil Android que hemos desarrollado en este proyecto:

- **Hilo de captura de imágenes.** Su misión es realizar captura de imágenes en segundo plano, incluso con la aplicación minimizada, mientras realizamos otras tareas o tenemos el teléfono bloqueado con la pantalla inactiva. También recae sobre este hilo la responsabilidad de realizar las conversiones necesarias a la imagen, aplicar la compresión y realizar el envío de las mismas.

- **Hilo de recepción de comandos.** La tarea que lleva acabo es la de recibir comandos desde el servidor y reaccionar ante dichos comandos, bien respondiendo información del estado del sistema o bien ejecutando acciones sobre la configuración de la cámara o el teléfono. Su ejecución es en paralelo con el hilo de captura de imágenes. Cuando se recibe un comando de petición de imagen de un usuario, se comunica con el hilo de captura de imágenes para que realice el envío de la última imagen disponible. Se puede ver el flujo de ejecución de ambos hilos en la imagen de la Figura 4.1.

- **Servicio.** La aplicación lanza un servicio en segundo plano que permite que el hilo de captura y el de recepción de comandos puedan seguir funcionando correctamente incluso con la interfaz gráfica cerrada, ya que deja un proceso en

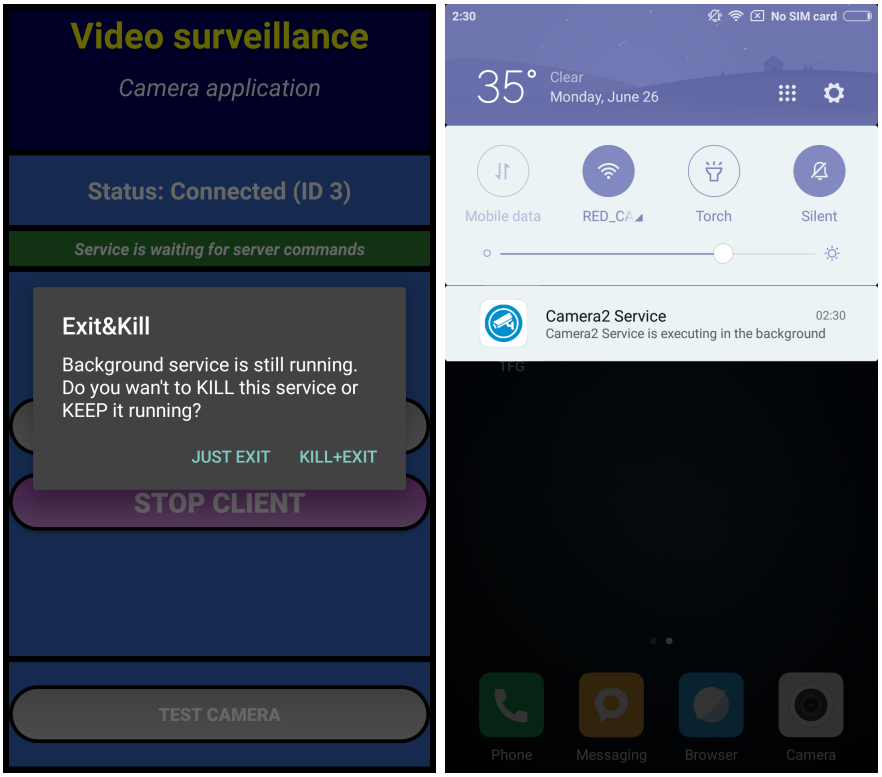

Figura 4.2: Servicio en segundo plano

segundo plano que seguirá ejecutando dichos hilos. Para que Android no termine automáticamente el proceso después de cierto tiempo (suele ser alrededor de un minuto), se necesita activar una notificación fija en el área que tiene Android diseñada para ello (parte superior). Podemos ver cómo quedaría el área de notificaciones en la Figura 4.2.

- **API.** Se dispone de una interfaz de programación para simplificar el uso de funcionalidades que nos ofrece la API de Android. Se ha dividido en diferentes paquetes según la funcionalidad ofrecida:

  1. Consultar o modificar los parámetros de conectividad wifi o bluetooth.
  2. Configurar la captura (resolución, FPS, detección de personas, calidad de compresión, etc), empezarla, detenerla o reiniciarla con los nuevos valores.
  3. Consultar (o modificar en un futuro, ver capítulo 6) parámetros de la CPU, así como el número de núcleos activos.
  4. Convertir imágenes a otros formatos y comprimirlas en JPEG (Figura 4.1) usando las librerías ofrecidas por OpenCV (Anexo A). También puede

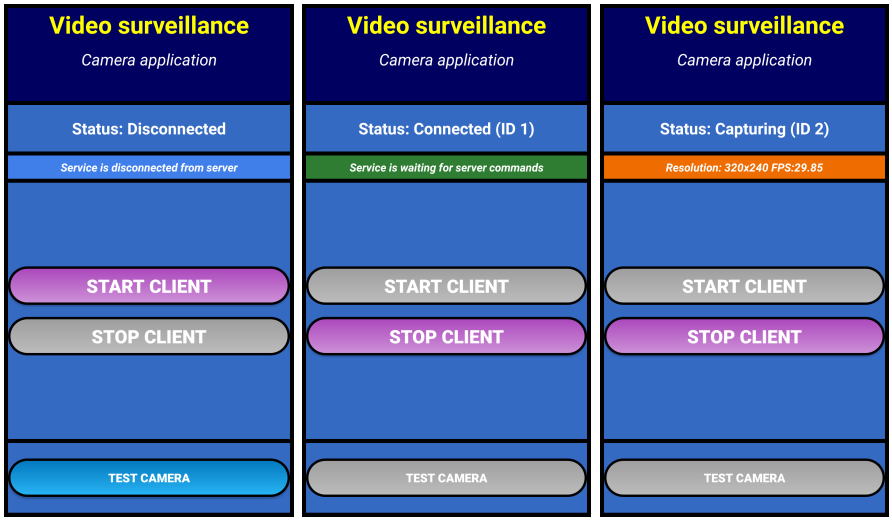

Figura 4.3: Interfaz gráfica en Android

aplicar algoritmos de detección de personas sobre las imágenes procesadas.

- **Interfaz gráfica**. Tenemos también una parte dedicada a la interfaz gráfica de usuario, que nos permite además una funcionalidad muy básica para poder seleccionar las opciones de captura y conexiones iniciales de forma visual y rápida (Figura 5.1). Dispone de un hilo que comprueba cada 200ms el estado de la conexión con el servidor y los FPS a los que se está capturando y procesando (Figura 4.3). La interfaz es adaptable a cambios de orientación en el teléfono, es decir, se renueva cuando el teléfono pasa de una posición vertical a horizontal y viceversa. Para desarrollar la interfaz gráfica se deben definir los diferentes elementos en una serie de ficheros xml con una sintaxis y un orden especificado por Android.

4.3.2. Servidor

La aplicación servidor (Figura 4.4) es el corazón del sistema. El servidor es el encargado de realizar la gestión de múltiples cámaras, interactuar con la aplicación cliente y hacer comprobaciones en la integridad de cada comando para ganar fiabilidad y robustez. Es el módulo responsable de la estabilidad de todo el sistema, ya que todas las comunicaciones tienen que pasar por él. Si el servidor cayese, todo el sistema caería. Por lo que las pruebas de disponibilidad en el capítulo 5 se hacen fundamentales.

Los componentes por los que está formado el servidor son los siguientes:

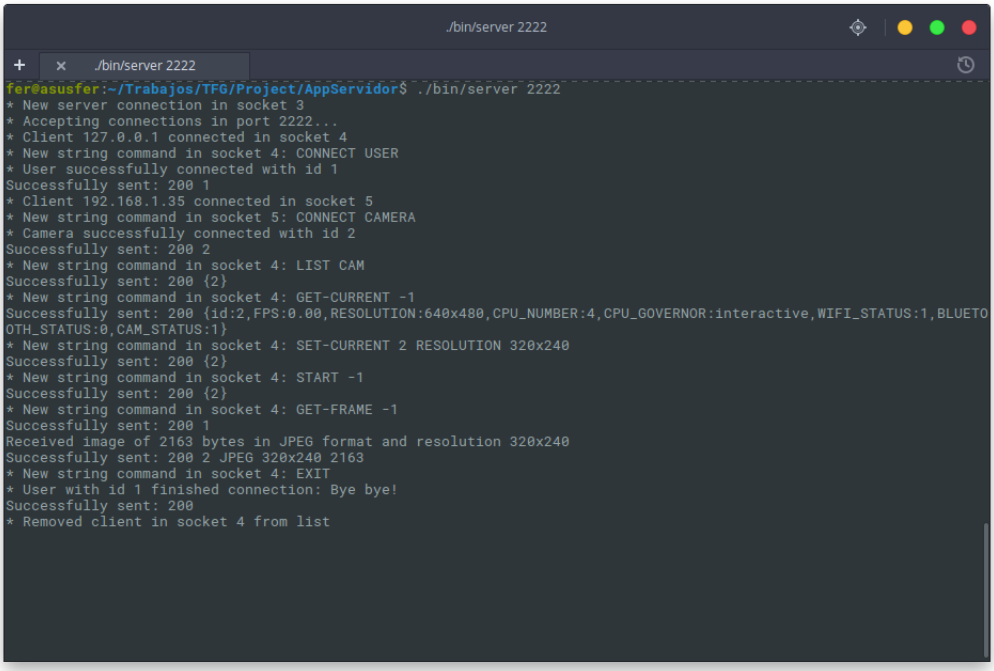

Figura 4.4: Servidor en funcionamiento

- **Server**. Se encarga de la apertura de un socket para el servidor y aceptar nuevos clientes para mandarlos a hilos de ejecución paralelos.

- **Thread**. Es la parte responsable de la gestión de la conexión con cada cámara o usuario de sistema. Se diferencia entre cámaras o usuarios mediante un parámetro en el comando connect que nos permite diferenciar de cual se trata. Este componente permanece a la espera de comandos por parte de los usuarios y en el caso de las cámaras se asegura de que siguen en línea y les manda acciones cuando sean requeridas por algún usuario.

- **Command**. Esta parte es la encargada del parseo, ejecución y envío de comandos. Aquí es donde se aplica el protocolo de comunicaciones definido de forma estricta para poder lograr una comunicación satisfactoria en todas las conexiones.

- **Lista de conexiones**. Siempre se mantiene una lista de estructuras con información acerca de cada conexión que nos permitirá saber en cada conexión socket si está establecida con un usuario o una cámara, la dirección IP a la que pertenece, etc. Esta estructura es fundamental para poder gestionar múltiples cámaras y/o aplicaciones clientes de forma simultanea.

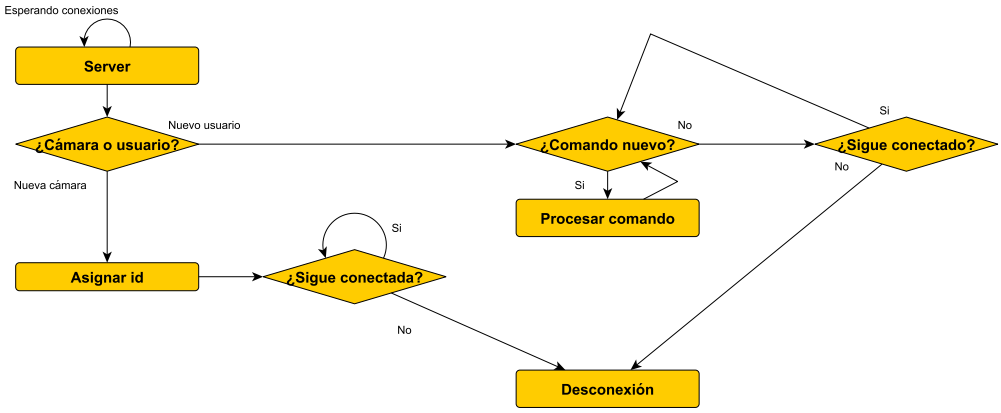

Figura 4.5: Diagrama de flujo del servidor

Se puede apreciar un diagrama de flujo de la aplicación servidor en la Figura 4.5.

4.3.3. Aplicación cliente

Se ha desarrollado una aplicación cliente para poder gestionar las cámaras de forma remota sabiendo únicamente la IP y el puerto donde esté ejecutándose el servidor. Consiste en una aplicación de consola que será manejada por el usuario encargado de mandar los comandos que necesite.

Las acciones que se pueden realizar (visibles en la Figura 4.6) desde esta aplicación son las siguientes:

1. **Listar cámaras.** Este comando obtiene la lista de cámaras conectadas al servidor.

2. **Get current.** Este comando nos permite obtener el valor de diferentes parámetros de captura y estados de las cámaras. Es un comando flexible que nos permite la posibilidad de:

   - Obtener un parámetro o estado concreto de una cámara específica.
   - Obtener un parámetro o estado concreto de todas las cámaras conectadas al servidor.
   - Obtener todos los parámetros y estados de una cámara específica.
   - Obtener todos los parámetros y estados de todas las cámaras conectadas al servidor.

3. **Set current.** Este comando va de la mano con el anterior. Permite configurar

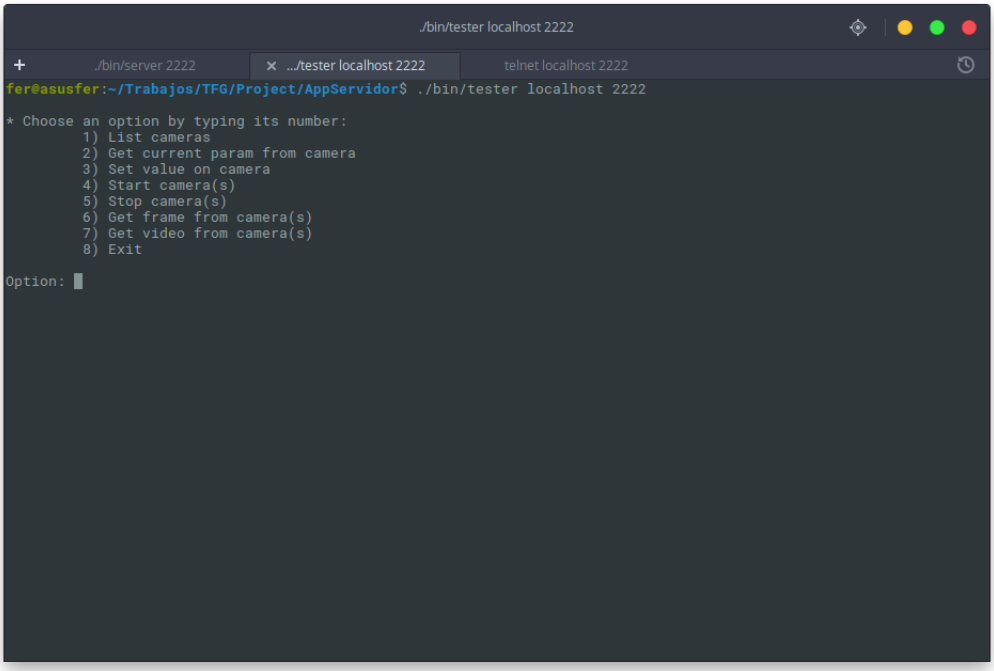

Figura 4.6: Aplicación cliente

los parámetros de captura de una o todas las cámaras, pero tenemos que hacerlo de uno en uno.

4. **Start.** Este comando provoca que el servidor ponga en estado de captura a una cámara en específico o todas ellas. Esto significa que las cámaras comenzarán a capturar imágenes y procesarlas pero aún no las enviarán hasta que se lo pidamos con el comando Get frame.

5. **Stop.** Este comando provoca que el servidor detenga la captura de imágenes en una o todas las cámaras conectadas al servidor. Esto provocará que las cámaras que estén en estado de captura cambien a un estado de inactividad, ahorrando batería si no es necesario que inicien la captura aún.

6. **Get frame.** Obtiene una imagen de una o todas las cámaras que estén conectadas al servidor y se encuentren en estado de captura. Esta imagen es mostrada por OpenCV en una nueva ventana.

7. **Get video.** Realiza peticiones a get frame en bucle. De esta forma conseguiremos una sensación de movimiento y podremos visualizar lo que está pasando en las cámaras en tiempo real. Se trata de una primera versión de lo que en un

Algoritmo 4.1 Estados wifi simplificados en la API

```
public static final int WIFI_NOT_SUPPORTED = -1;
public static final int WIFI_DISABLED =  0;
public static final int WIFI_ENALBED =  1;
```

Algoritmo 4.2 Método desarrollado para simplificar el Wifi

```
public static int isWifiEnabled( Context context) {
    android.net.wifi.WifiManager wifi = (android.net.wifi.WifiManager) context.getApp

    if( wifi == null) {
        return WIFI_NOT_SUPPORTED;
    }

    return wifi.isWifiEnabled() ? WIFI_ENALBED : WIFI_DISABLED;
}
```

futuro podría ser implementado como un comando independiente más eficiente como hablaremos en los capítulos 5 y 6.

8. **Exit.** Abandona la aplicación cliente y termina la conexión con el servidor.

4.4. API para la funcionalidad en Android

Tenemos diferentes funcionalidades Android que nos permiten realizar acciones complejas en unas pocas líneas de código. Como estas funcionalidades son muy utilizadas en la aplicación, se ha decidido realizar una API para acceder a esas funcionalidades de una manera eficiente y robusta. Las funcionalidades que se incluyen en esta API son las de:

- Activar/Desactivar la conexión Wifi. Esta opción nos permite activar o desactivar el wifi de forma remota (ver Algoritmo 4.1 y Algoritmo 4.2).

- Activar/Desactivar la conexión Bluetooth. Esta opción nos permite activar o desactivar el bluetooth de forma remota.

- Configurar los parámetros de captura. Este método nos permite configurar diferentes opciones de captura de imágenes como su resolución, FPS limitado por software (ya que en Android no existe esa posibilidad por hardware), autoenfoque, flash, etc.

- Conversión de imagen en formato YUV a RGB (Algoritmo 4.3) y compresión RGB a JPEG.

Algoritmo 4.3 Conversión de YUV a RGB

```
Imgproc.cvtColor( yuvMat, rgbMat, Imgproc.COLOR_YUV420sp2RGB, 3);
```

Algoritmo 4.4 Algoritmo de detección de personas

```
/** @return Number of people detected */
static int detectPeople(Mat img) {
    HOGDescriptor hog = new HOGDescriptor();
    hog.setSVMDetector(HOGDescriptor.getDefaultPeopleDetector());

    final MatOfRect foundLocations = new MatOfRect();
    final MatOfDouble foundWeights = new MatOfDouble();

    final Size winStride = new Size(8, 8);
    final Size padding = new Size(32, 32);

    hog.detectMultiScale(img, foundLocations, foundWeights, 0.0, winStride, paddi
    Rect[] array = foundLocations.toArray();

    int thickness = 5; //espesor rectangulo

    for(Rect r: array) {
        Scalar color = new Scalar(Color.red(rectangleColor), Color.green(rectangl
        Imgproc.rectangle(img, new Point(r.x, r.y), new Point(r.x+r.width, r.y+r.
    }

    return array.length;
}
```

- Aplicación del algoritmo de detección de personas de OpenCV (Anexo A y Algoritmo 4.4).

4.5. Protocolo de comunicaciones del sistema

Para realizar una comunicación exitosa entre todos los módulos que componen el sistema es necesario diseñar e implementar un protocolo de comunicaciones eficiente, robusto y coherente. En la Figura 4.7 tenemos un diagrama explicativo del flujo de las comunicaciones entre distintos módulos que varía según el tipo de comando a ejecutar.

Una vez entendido esto, procedemos a la definición de los comandos que permitan realizar las acciones solicitadas. En la tabla 4.1 se muestra una tabla con los comandos existentes dentro del sistema con su funcionalidad y sintaxis detallada.

4.6. Aplicaciones prácticas

En esta sección vamos a hablar de las diferentes aplicaciones que actualmente se le dan al proyecto. Hablaremos de su utilidad como sistema de videovigilancia en

	Parámetros	**Utilidad**
Connect	Cam/User	Conecta una cámara o un usuario al servidor, que le asignará un identificador único
List	Cam/Cmd	El servidor devuelve la lista de cámaras conectadas o la lista de todos los comandos disponibles
Get-current	ID param	Obtiene el valor de un parámetro de una cámara. Puede ser utilizado para devolver todos los parámetros de una cámara o de todas también
Set-current	ID param value	Configura un parámetro concreto para una de las cámaras o todas las conectadas
Start	ID	Comienza a capturar imágenes en una cámara o en todas las que estén inactivas
Stop	ID	Detiene la captura de imágenes en una cámara o en todas las que estén capturando
Get-frame	ID	Obtiene la última imagen disponible en la cámara especificada o en todas
Exit		Termina la conexión con el servidor
Quit		Este comando solo se utiliza en ocasiones que deseamos hacer pruebas y no se accede desde ninguna aplicación. Desasigna el identificador asociado a la conexión pero no cierra la conexión con el servidor. Puede permitir a un usuario dejar de ser usuario y pasar a ser cámara.

Tabla 4.1: Sintaxis de los comandos del protocolo

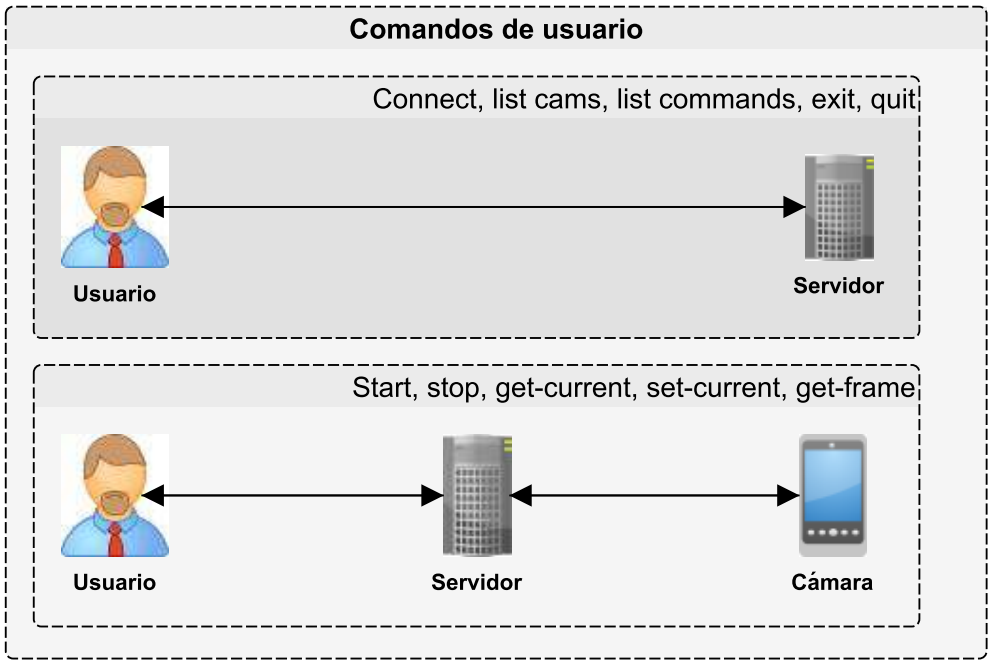

Figura 4.7: Diagrama de comunicaciones del usuario

tiempo real, que es el principal objetivo para el que fue diseñado y posteriormente se describe su utilidad como alarma de detección de personas de forma remota.

4.6.1. Videovigilancia en tiempo real

La principal aplicación practica, y la más obvia, es la de diseñar un sistema de videovigilancia compuesto por smartphones y un servidor al que envíen las imágenes. Como ya se habló en apartados anteriores, el hecho de que las cámaras sean teléfonos Android nos permite mayor flexibilidad que los típicos sistemas de videovigilancia que necesitan de un hardware concreto para funcionar.

En la figura 4.8, tenemos una fotografía del sistema de videovigilancia en funcionamiento con cuatro cámaras puestas en posiciones estáticas ancladas a unos trípodes.

Es un sistema ideal para este propósito, ya que se desarrollo pensando en esta aplicación como objetivo principal y la forma en la que está diseñado el sistema lo favorece. Siempre es posible mejorarlo y sobretodo mejorar la resolución, e imágenes por segundo obtenidas, como hablaremos en los capítulos 5 y 6.

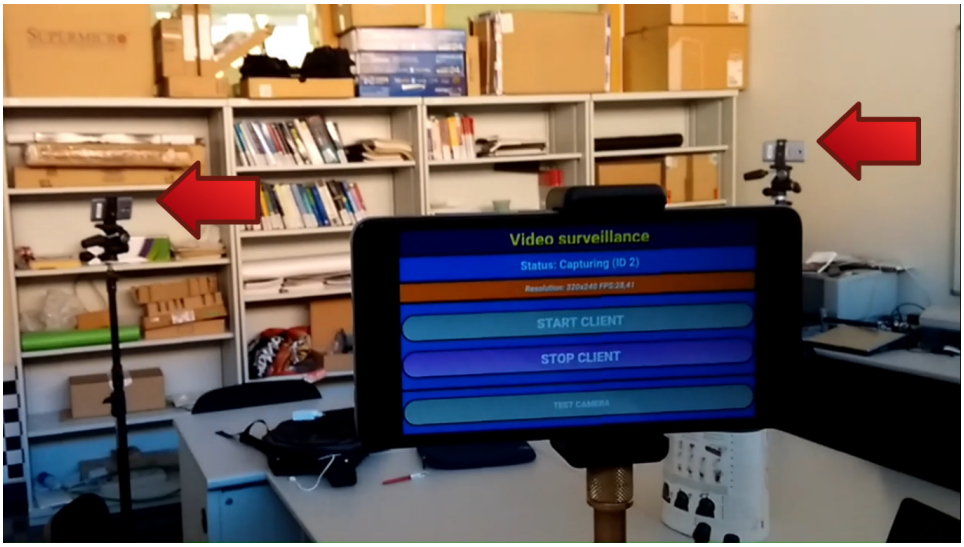

Figura 4.8: Sistema de videovigilancia usando trípodes

4.6.2. Detección de personas de forma remota

Disponemos de la opción de enviar las imágenes solo cuando se detecte alguna persona en la escena. Esto nos facilita la opción de utilizar este sistema como una alarma a control remoto que nos informará de la situación cuando haya detectado alguna persona en la imagen. Dicha funcionalidad es aplicable a nivel profesional por empresas de seguridad privada que les facilitaría mucho el trabajo de descartar peligro cuando alguna alarma saltase para confirmar con las imágenes recibidas junto a la alarma si la amenaza es real o no. En la figura 4.9 tenemos un ejemplo de cómo el sistema resalta con un recuadro de color verde (este color es configurable) cuando la opción de detección de personas es activada. En esta imagen se puede observar como se distingue la persona del entorno aplicando el algoritmo de detección de personas que está integrado en la herramienta OpenCV (ver anexo A).

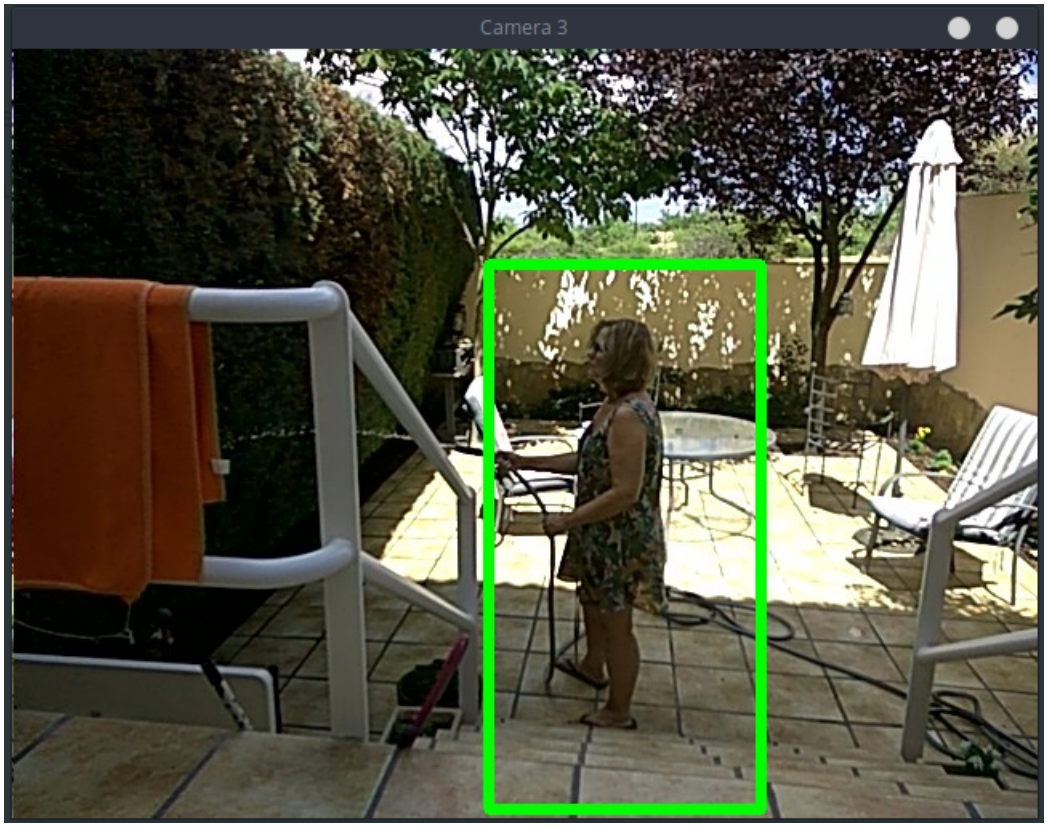

Figura 4.9: Ejemplo de detección de personas

Capítulo 5

Trabajo experimental

Este capítulo está dedicado a la realización de experimentos con diferentes parámetros de captura, diferente número de cámaras conectadas, variaciones en la resolución, etc. Se realizarán mediciones para determinar los cuellos de botella existentes en un entorno experimental lo más real posible y se sacarán una serie de valiosas conclusiones útiles para el futuro de este proyecto como se reflejará en el capítulo 6.

5.1. Requisitos de instalación del entorno

Para poder ejecutar el sistema correctamente necesitamos una preparación del entorno previa. En esta sección se va a explicar cuales son los requisitos para poder realizar una integración exitosa del sistema desde el punto de vista del usuario.

Para empezar será necesario que el usuario descargue el ejecutable para Android que le permita instalar la aplicación. En este caso, se trata de un archivo instalador (apk) que realizará la instalación de la misma de forma automática. Es una aplicación algo pesada (aproximadamente 40MB a día de hoy), esto es debido a que se decidieron integrar las librerías de OpenCV para todas las plataformas disponibles de modo que en tiempo de ejecución se cargue la librería necesaria sin necesidad de instalar dependencias adicionales a la propia app. Esto se realizó así para asegurar siempre que la versión de OpenCV que se tiene es la misma con la que se realizan las pruebas, para asegurarse de tener siempre la versión más nueva y optimizada de OpenCV y por facilidad de instalación por parte de un usuario sin demasiados conocimientos.

Una vez instalada la app en Android, debemos estar en una red con acceso a la IP y puerto del servidor. Para ello es necesario conocer previamente en qué ordenador se ha instalado la aplicación servidor de la que hablaremos más adelante en esta misma sección. Como se muestra en la Figura 5.1, debemos configurar algunos campos para

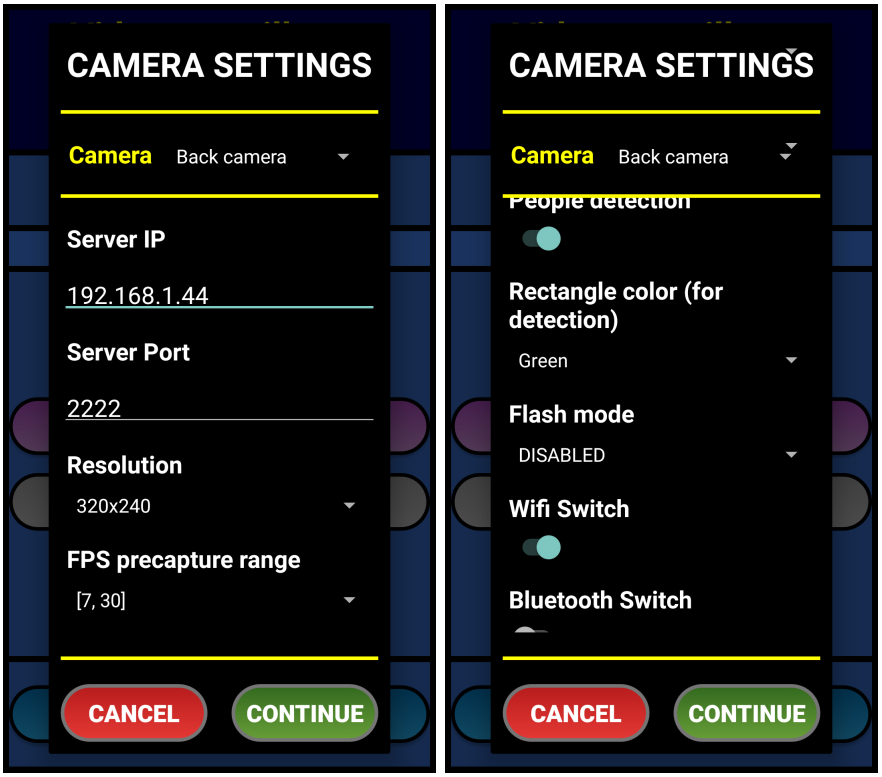

Figura 5.1: Configurando la cámara inicialmente

poder conectarnos al servidor con unos parámetros por defecto que se podrán cambiar desde la aplicación de usuario.

El servidor al que se conectan las cámaras debe ser un servidor con suficiente capacidad de procesamiento, una cantidad suficiente de memoria RAM, ancho de banda en la red para poder mantener un streaming de vídeo de alta calidad multicámara. Todos estos requerimientos dependen de los objetivos que nos marquemos como se podrá analizar en las pruebas de rendimiento de las siguientes secciones. Una vez nos hemos asegurado de que el ordenador cumple con los mínimos requisitos hardware, debemos instalar una distribución de linux estable (como por ejemplo Debian o CentOS). Y ahora, una vez tenemos el sistema operativo correctamente instalado y actualizado, debemos ejecutar la aplicación servidor en un terminal para que pueda recibir conexiones de clientes, que serán cámaras o usuarios indistintamente.

Por último, para poder enviar comandos a las cámaras a través del servidor ya instalado, debemos tener otro ordenador con linux con las librerías de OpenCV instaladas en él, que se pueden obtener de prácticamente todos los repositorios de las principales distribuciones linux de usuario (como Ubuntu y derivados). Este ordenador debe tener acceso en red a la ubicación física del servidor o también podemos

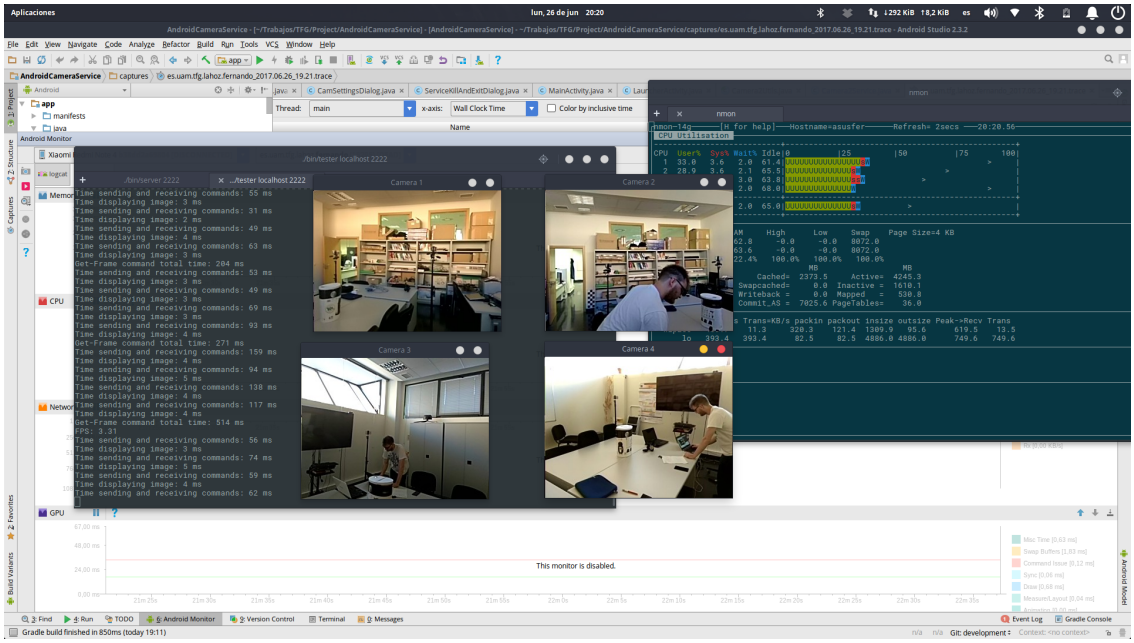

Figura 5.2: PC con cliente, servidor y monitorización

utilizar el mismo PC que para el servidor indicando siempre al cliente que la IP sería la propia y el puerto concreto que se haya decidido para el servidor anteriormente.

5.2. Medición de recursos

En todas las pruebas realizadas en este capítulo, se ha optado por instalar juntos en el mismo PC el servidor y la aplicación cliente. (Figura 5.2). Cuando en las mediciones efectuadas se hace referencia al PC se refiere a este ordenador, con ambas aplicaciones instaladas (Tablas 5.2, 5.3, 5.4). Los datos reflejados en dichas tablas son los resultados de la media aritmética de las mediciones efectuadas.

Para realizar dichas mediciones, se ha utilizado una combinación de la herramienta de monitorización para linux *nmon* y una herramienta de monitorización de distintos parámetros que nos ofrece el IDE de Android Studio (ver Figura 5.3).

Para realizar todos los experimentos hemos construido un entorno de pruebas lo más parecido a la realidad, en el que hemos dispuesto cuatro cámaras en cuatro posiciones fijas sobre trípodes en diferentes ángulos, apuntando al centro de una habitación en la que pusimos un objeto como referencia (Figura 4.8).

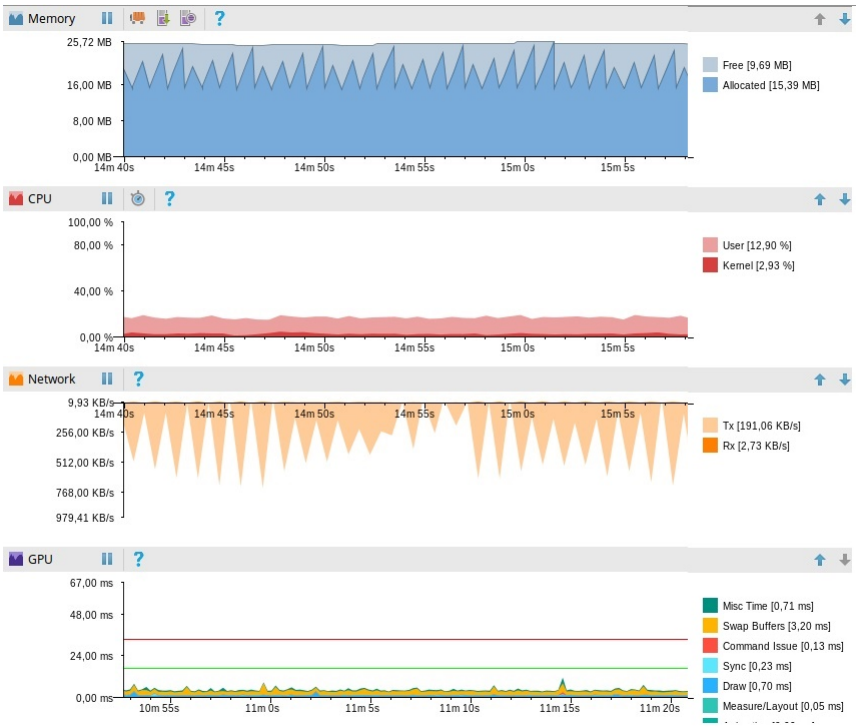

Figura 5.3: Monitorización en Android Studio

5.3. Pruebas de disponibilidad

Como se menciono en el capítulo 3, la disponibilidad del sistema es uno de los requisitos no funcionales más importantes. Para comprobar que dicho requisito se cumple, se realizarán una serie de pruebas genéricas para comprobar el funcionamiento estable y sin caídas de cada uno de los módulos de la aplicación.

Bajo las condiciones explicadas en la sección 5.2 se realizaron experimentos para garantizar que el servidor y la aplicación cliente reaccionaban adecuadamente ante cambios de conexión en las cámaras, desconexión repentina del servidor en un momento al azar (ver Figura 5.4) y flujos de ejecución complejos donde se realizaban transmisiones de vídeo en tiempo real desde una, dos y cuatro cámaras simultáneamente. En todas estas ocasiones los resultados fueron los esperados y el sistema demostró ser estable y robusto frente a fallos. Esto se puede confirmar con las pruebas realizadas de disponibilidad del sistema (Tabla 5.1) en las que dejamos el sistema con varias cámaras en funcionamiento durante varias horas sin problemas más allá de la descarga de la batería del teléfono móvil.

Nº cámaras	Tiempo operando	Tiempo disponible
1	1 minuto	100 %
1	1 hora	100 %
1	8 horas	100 %
2	1 minuto	100 %
2	1 hora	100 %
2	8 horas	100 %
4	1 minuto	100 %
4	1 hora	100 %
4	8 horas	100 %

Tabla 5.1: Experimento de disponibilidad del servidor

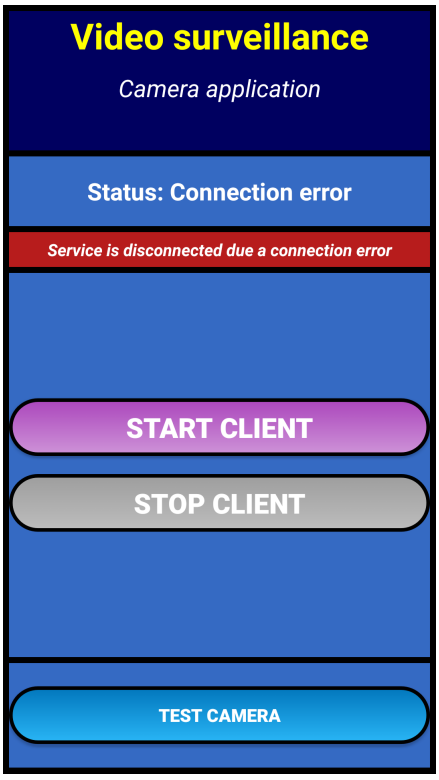

Figura 5.4: Desconexión repentina del servidor

	320x240		640x480		1280x720	
	PC	**Cámara**	**PC**	**Cámara**	**PC**	**Cámara**
Saturación de la red	287KB/s	290KB/s	539KB/s	509KB/s	355KB/s	330KB/s
Carga de CPU	40 %	16 %	40 %	36 %	35 %	90 %
FPS	20	30	12	27	4	10

Tabla 5.2: Mediciones con una cámara y un PC

5.4. Pruebas de rendimiento

En esta sección vamos a realizar una serie de experimentos para comprobar el comportamiento del sistema en diferentes situaciones. Se analizarán situaciones en las que variaremos la resolución con una, dos o cuatro cámaras conectadas para comprobar cuánto varían las imágenes por segundo que recibirá el usuario, así como la carga de CPU, saturación de la red y uso de memoria RAM.

Todas las pruebas han sido realizadas utilizando un mismo PC para ejecutar simultáneamente servidor y aplicación cliente, mientras que los smartphones estaban conectados a él a través de la misma red local wifi, dedicando un router exclusivamente para este cometido.

5.4.1. Rendimiento con una cámara

A continuación, mostramos una tabla con los resultados de la variación de la resolución de captura frente a los FPS, carga de CPU y saturación de la red obtenidos en dos sitios: la app del smartphone y la aplicación de usuario en el PC. Esto se mide en dos sitios porque no es lo mismo la frecuencia con la que es capaz de emitir las imágenes y procesarlas el teléfono a la frecuencia con la que estas imágenes acaban llegando al usuario. Esto es debido a que el diseño de la aplicación cliente no está pensado para gestionar múltiples transmisiones de vídeo simultáneamente y el servidor no dispone de ningún buffer de almacenamiento de imágenes temporal. Estos son dos puntos muy importantes a mejorar de cara al trabajo futuro como se habla en el capítulo número 6.

Como podemos observar en la Tabla 5.2, a altas resoluciones, los FPS se ven afectados de forma significativa debido a que el número de píxeles a procesar aumenta considerablemente y esto afecta en el proceso de conversión del espacio de color (YUV a RGB), la compresión (RGB a JPEG) y en su envío y recepción, reflejándose en el aumento de la carga de la CPU.

La cantidad de RAM requerida por el sistema apenas varía en los diferentes experimentos como pudimos observar con la herramienta de monitorización de Android Studio (Figura 5.3). Esto es debido a que no hay buffers intermedios en ninguno de los módulos del sistema que almacenen imágenes. Por lo tanto, desde el punto de vista de la memoria RAM, el uso se puede decir que es constante y ligero. No se han anotado las mediciones en las tablas ya que este parámetro no sufría variaciones significativas, como era de esperar.

Para medir la saturación de la red, se ha utilizado el indicador preinstalado de Android en los smartphones y un el programa de consola nmon para el PC con Linux (Figura 5.2). Es importante recalcar la importancia de controlar la saturación de la red ya que si se envían imágenes en bruto por la red (por ejemplo en formato YUV420) el uso de red es mucho mayor que en un formato comprimido como JPEG, con una calidad de un 90 % aproximadamente, que no hay una perdida significativa de calidad y, sin embargo, el ahorro en bytes a la hora de transmitirlo en tiempo real es muy alto a cambio de unos pocos mili-segundos a la hora de comprimir la imagen antes de enviarla. Este parámetro depende de muchos factores como el ritmo de envío de imágenes en las cámaras, resolución de las imágenes, calidad de la compresión JPEG, etc. Por eso es muy útil para comprobar las tablas y ver donde están los cuellos de botella del sistema, que serán las partes más saturadas.En este caso, se ve como aumenta la saturación de la red con la resolución intermedia (640x480) pero luego vuelve a bajar para la más alta (1280x720), ya que la cantidad de imágenes por segundo que está enviando la cámara a esa resolución es de tan solo un tercio de lo que era antes.

5.4.2. Rendimiento multicámara

En este apartado se van a realizar mediciones sobre los valores de FPS, carga de CPU y saturación de red que se consiguen en el sistema con varias cámaras conectadas de forma simultánea y distintas resoluciones. De esta forma podemos medir de forma científica en qué grado afecta al sistema operar en modo multicámara. Para ello, hemos realizado algunas mediciones utilizando las mismas herramientas que en el apartado anterior (ver Figura 5.3 y Figura 5.3) y la información sobre las imágenes por segundo ofrecidas por la aplicación cliente y la app Android.

En la Figura 5.3 se puede apreciar como se reduce el tráfico de red con respecto a las mediciones realizadas con una cámara, y los FPS en el cliente han bajado en torno a la mitad. Esto es debido a la limitación de diseño ya comentada que sufre la aplicación cliente. No está diseñado para gestionar múltiples transmisiones de vídeo simultáneas.

	320x240		640x480		1280x720	
	PC	**Cámara**	**PC**	**Cámara**	**PC**	**Cámara**
Saturación de la red	340KB/s	155KB/s	522KB/s	250KB/s	456KB/s	200KB/s
Carga de CPU	40 %	16 %	45 %	36 %	38 %	90 %
FPS	11	30	7	28	2	10

Tabla 5.3: Mediciones con dos cámaras y un PC

	320x240		640x480		1280x720	
	PC	**Cámara**	**PC**	**Cámara**	**PC**	**Cámara**
Saturación de la red	412KB/s	100KB/s	357KB/s	95KB/s	458KB/s	150KB/s
Carga de CPU	40 %	16 %	40 %	36 %	35 %	90 %
FPS	4	30	1	30	0,82	10

Tabla 5.4: Mediciones con cuatro cámaras y un PC

Con respecto a la memoria RAM ocurre lo mismo que cuando tenemos una sola cámara, permanece estable.

En la tabla de la Figura 5.4 se puede observar como sigue cayendo el tráfico de red en Android significativamente, ya que las peticiones del usuario a las cámaras se realizan con un intervalo mayor de tiempo al haber múltiples clientes atendidos de forma secuencial. Esto afecta directamente a los FPS que verá el usuario, que ahora son de menos del 25 % de los obtenidos en los experimentos con una cámara (ver Tabla 5.2).

Como podemos observar en las Figuras 5.3 y 5.4, cuanto mayor es el número de cámaras conectadas menores son los FPS que conseguimos en la aplicación cliente del PC. Esto es principalmente debido a como está implementada dicha aplicación cliente, ya que se podría mejorar de forma notable algunos aspectos como son el hecho de tener que estar enviando un comando a la cámara cada vez que se requiere una imagen, el hecho de no estar procesando cada cámara en un hilo independiente y la ausencia de un buffer de almacenamiento intermedio en el servidor. En el capítulo número 6 se hablará sobre ello en detalle. Además podemos notar que en todos los casos es significativa la caída de FPS en Android cuando el sistema funciona a altas resoluciones, ya que el tiempo que tardará en realizar la transformación de la imagen al formato deseado será más costosa, sobrecargando la CPU de la cámara con dichas operaciones.

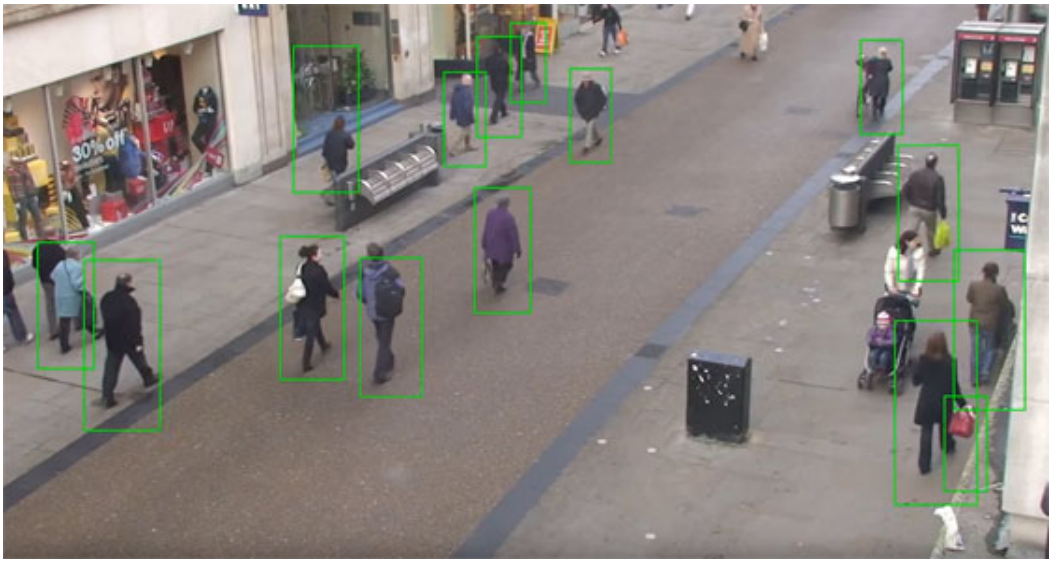

Figura 5.5: Algoritmo de detección de personas de OpenCV en un entorno real

Con respecto a la memoria RAM ocurre lo mismo que cuando tenemos una sola cámara, permanece estable.

5.5. Ejemplo práctico: detección de personas

Hemos realizado una serie de pruebas con un sistema de cuatro cámaras conectadas a una misma red local, a las que se accede mediante la aplicación de usuario instalada en un PC junto a la aplicación servidor. Activamos la detección de personas y comprobamos el funcionamiento del algoritmo para comprobar cómo los FPS caían, sobretodo a resoluciones altas, ya que al tratarse de un algoritmo complejo, el rendimiento de la aplicación Android es peor. Sin embargo, se consiguió tener el sistema funcionando con unos números de 640x480 de resolución y un FPS de aproximadamente 2 imágenes por segundo.

En la Figura 5.5 podemos ver como funciona un algoritmo de detección de personas de OpenCV (Anexo A).

5.6. Conclusiones

En los casos analizados anteriormente, el consumo de CPU por parte del PC con las aplicaciones cliente y servidor fue más o menos constante en todo el experimento, mientras que en la parte de la aplicación móvil se incrementaba significativamente con el aumento de la resolución a procesar.

El peor de los casos para el consumo de CPU es el de transmisión de vídeo a una cantidad alta de imágenes por segundo y resoluciones altas. Si las cámaras Android fueran capaces de realizar el procesamiento de las imágenes a alta velocidad, siendo constantes los FPS en el máximo definido por hardware (30 en los teléfonos con los que se realizaron las pruebas), tendríamos mucha más sobrecarga en el PC con las aplicaciones servidor y cliente, ya que tendría que visualizar múltiples transmisiones de alta calidad en intervalos cortos de tiempo debido al alto valor teórico del FPS. Como esto no es así y las cámaras suponen el primer cuello de botella, la carga de la CPU en el PC permanece prácticamente constante, notando un pequeño incremento en el caso de dos cámaras con 640x480 y 30 FPS en las cámaras, lo cuál apoya esta teoría.

Los parámetros que más afectan al rendimiento del sistema son la resolución de captura y el número de cámaras conectadas. Como hemos podido ver en los apartados anteriores el cuello de botella está en el tiempo de procesamiento, que no es suficientemente veloz para poder tener sensación de movimiento fluido con resoluciones altas. Esta sensación se produce a partir de unas 18 imágenes por segundo aproximadamente (aunque la discusión sobre este número es un tema amplio que no abarcaremos aquí), es decir, que como máximo el procesamiento de las imágenes no puede durar más de 55 mili-segundos.

En el siguiente capítulo nos centraremos en mejoras específicas que mejorarían el rendimiento para permitir una grabación a mayores resoluciones y mayor número de imágenes por segundo.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

En este proyecto presentamos un sistema de videovigilancia utilizando smartphones como cámaras, y se han cumplido con los requisitos que se mencionaron. Las funcionalidades que nos ofrece un sistema operativo como Android en cualquier teléfono inteligente moderno son inmensas y queda demostrado que se le puede sacar provecho con un proyecto como este. La cantidad de sensores disponibles, la cámara (cada vez mejores) que suele venir integrada en el teléfono, el micrófono, etc, nos permite desarrollar aplicaciones sobre diversos temas, como por ejemplo la videovigilancia.

El desarrollo de aplicaciones para dispositivos móviles está en auge, no es ningún secreto, y sin embargo se han desarrollado muy pocos sistemas que vean a los teléfonos inteligentes como una herramienta para obtener información útil a través de sus cámaras y sensores. Creo que este es un punto a explotar y que muy probablemente en el futuro se empiece a ver más sistemas que utilicen smartphones como dispositivos adaptables a casi cualquier entorno. La versatilidad que nos ofrece un smartphone a la hora de su uso no es comparable con los sistemas que se desarrollaban hace no tantos años en los que se necesitaba desarrollar hardware específico o comprar añadidos para poder realizar tareas que ahora mismo cualquier teléfono con el software adecuado es capaz de hacer.

El sistema que he desarrollado a lo largo de este último año es un sistema de videovigilancia completamente funcional y está en condiciones de competir con los sistemas caseros y algunos de los profesionales disponibles en Internet a día de hoy. El rendimiento del sistema es mejorable, y las medidas que habría que tomar para mejorar el sistema han sido analizadas y estudiadas.

La conclusión final que me queda es que este tema es muy amplio y siempre se puede seguir mejorando. Los sistemas de videovigilancia son complejos y están compuestos por múltiples capas, por lo que al final necesitas tener conocimientos sobre diversos campos como el renderizado de imágenes, funcionamiento de las cámaras, el rendimiento de los diferentes algoritmos de compresión, el consumo de ancho de banda de las transmisiones de vídeo, realizar mediciones precisas de tiempos, buffering, etc. Realizando el desarrollo de este proyecto, hemos ido anotando muchas posibles mejoras que habría que estudiar en un futuro, ya que cuando un tema es tan amplio, cualquier cambio por ligero que sea implica un estudio de los posibles caminos a tomar para lograr llevar a cabo los objetivos y eso suele ser costoso.

6.2. Trabajo futuro

A la vista de los resultados de las mediciones realizadas en los experimentos del capítulo número 5 hemos anotado algunas mejoras que quedan pendientes de realizar en un futuro para así mejorar la transmisión de vídeo en tiempo real en cuanto a calidad de las imágenes transmitidas, el ritmo de imágenes por segundo mostradas al usuario, la interacción gráfica con el usuario y la funcionalidad multicámara. A continuación, una lista de las mejoras que se podrían llevar a cabo para mejorar el proyecto:

- **Mejora de usabilidad en la aplicación cliente.** Se debería mejorar la interacción gráfica con el usuario. Convertirla en una aplicación para móvil y desarrollar una versión en Java multiplataforma sería muy buenas ideas para simplificar esta tarea de hacer una aplicación portable, ligera y amigable para el usuario, no como la que ahora mismo existe que es de consola y no tiene tanta usabilidad.

- **Añadir parámetros para poder programar horarios de captura.** Una interesante implementación en el futuro sería añadir la posibilidad de configurar horarios de captura de las cámaras para ahorrar energía el tiempo que no sea necesaria la videovigilancia. De este modo se ahorraría energía durante los intervalos horarios que no estuviera activo el sistema.

- **Mejorar el servidor para incluir semáforos.** Mejorar la aplicación servidor con la inclusión de semáforos para las estructuras que pueden ser accedidas por múltiples hilos correspondientes con conexiones con usuarios de la aplicación cliente. De este modo se conseguiría un servidor más robusto.

- **Poder definir más parámetros en Android.** Incluir la posibilidad de configurar la CPU para ahorrar batería cambiando el número de núcleos activos y sus frecuencias de funcionamiento[2] (con modo root en Android es posible).

- **Mejora de rendimiento en aplicación cliente.** Una mejora de la que ya se hablo en capítulos anteriores es la de crear un comando específico para vídeo, incluirlo en el protocolo de comunicaciones, y en vez de estar llamando en bucle a la función get frame utilizar una para vídeo que consista en una petición inicial a la cámara y el resto sería simplemente recibir mensajes por socket hasta que se quiera parar de recibir imágenes. Esto podría implementarse además de forma que cada grabación con cada cámara fuera gestionada en un hilo diferente mejorando enormemente los FPS logrados en la aplicación cliente y acercándose mucho a la limitación FPS hardware de la propia cámara.

- **Implementación de un buffer en el servidor.** La implementación de un buffer a nivel de servidor las imágenes fueran re-utilizables por varios usuarios simultáneos y de este modo una misma petición de imagen desde dos usuarios distintos sería contestada una sola vez por la cámara en vez de dos con dos imágenes diferentes como ocurre actualmente. Además la creación de un buffer en el servidor permitiría una reproducción de vídeo en el cliente mucho más estable porque la probabilidad de que haya jitter entre una imagen y otra se reduciría enormemente.

Bibliografía

[1] B. Chung, “People detection in opencv again,” 2011. 3

[2] K. Morell, “Governors en android, qué son y cuál es el mejor para ti,” 2014. 41

[3] OpenCV, “Android development with opencv,” 2011. 45

Apéndice A

Desarrollando en OpenCV

OpenCV (Figura A.1) es una biblioteca libre de visión artificial originalmente que fue originalmente desarrollada por Intel. Es utilizada en multitud de aplicaciones que necesitan hacer uso de algoritmos de detección de movimientos, reconocimiento facial y detección de personas (Figura A.2). También dispone de algoritmos de conversión de imágenes y de compresión.

En este proyecto usamos esta herramienta para llevar a cabo tareas de conversión de imágenes del formato nativo de la cámara Android YUV420sp a RGB para después comprimirlo en JPEG. Además se ha utilizado OpenCV para aplicar el algoritmo de detección de personas sobre la imagen RGB.

Para integrar OpenCV en Android basta con seguir los pasos de su web[3].

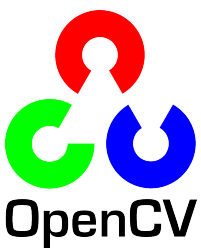

Figura A.1: OpenCV Logo

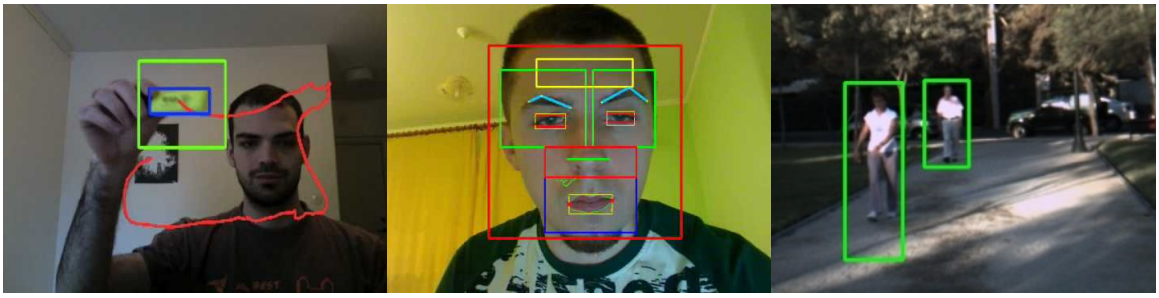

Figura A.2: OpenCV Algoritmos